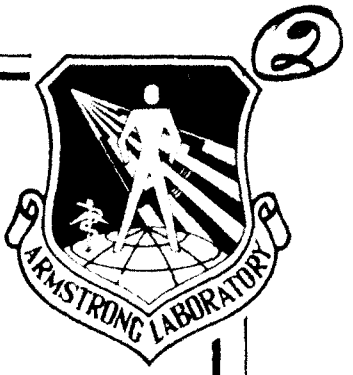
AL-TP-1992-0043



ARMSTRONG LABORATORY

DESIGN KNOWLEDGE MANAGEMENT SYSTEM (DKMS) BETA TEST REPORT

Richard J. Mayer
Thomas M. Blinn
David C. Browne
Matthew A. Grisius
Arthur A. Keen
Jeffery C. Lockledge
Les Sanders

KNOWLEDGE BASED SYSTEMS, INC.
2726 LONGMIRE
COLLEGE STATION, TX 77845-5424

HUMAN RESOURCES DIRECTORATE
LOGISTICS RESEARCH DIVISION

NOVEMBER 1992

INTERIM TECHNICAL PAPER FOR PERIOD NOVEMBER 1990 - JUNE 1992

AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6573

NOTICES

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation, or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this paper, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.

JOANN M. SARTOR
Program Manager

BERTRAM W. CREAM, Chief
Logistics Research Division

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1992	Interim - November 1990 - June 1992

4. TITLE AND SUBTITLE
Design Knowledge Management System (DKMS) Beta Test Report

5. FUNDING NUMBERS
C - F33615-90-C-0001
PE - 65502F
PR - 3005
TA - L2
WU - 03

6. AUTHOR(S)
Richard J. Mayer, Arthur A. Keen, Matthew A. Grisius, Thomas M. Blinn, Jeffery C. Lockledge, David C. Browne, Les Sanders

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Knowledge Based Systems, Inc.
2746 Longmire
College Station, TX 77845-5424

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Armstrong Laboratory
Human Resources Directorate
Logistics Research Division
Wright-Patterson AFB, OH 45433-6573

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AL-TP-1992-0043

11. SUPPLEMENTARY NOTES
Armstrong Laboratory Technical Monitor: Capt JoAnn Sartor, (513) 255-5775. This research was conducted under the Small Business Innovation Research (SBIR) Program as a Phase II effort

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Beta Test Report, prepared under a Phase II Small Business Innovation Research (SBIR) effort, documents potential use of the Design Knowledge Management System (DKMS) in research and industry. Industry concerns, gathered from surveys, are addressed in each subsection. The appendix gives an in-depth overview of the entire DKMS. The DKMS consists of: Container Object System: a flexible means of storing computer objects and constraints, so that a design can easily evolve as information changes. Model Design Support Environment: a method for automatic solution and code generation based on the problem, the "givens", and the real-world equations that rule the environment. High-Productivity CAD: a toolbox for CAD designers. Shape-based Design Knowledge Representation and Reasoning: component which allows shape information as a key factor in the design. (For example: "Does the shape of this part create unacceptable stress points?") Integration Platform: component which helps user select and translate services.

14. SUBJECT TERMS			15. NUMBER OF PAGES
artificial intelligence automatic code generation	CAD systems design *computer-aided design* concurrent engineering design	design evolution *SBIR program* shape-based reasoning	72 16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Table of Contents

Preface

This document provides the results of industry feedback with respect to the Design Knowledge Management System (DKMS) project, which was funded by the Armstrong Laboratory, Logistics Research Division, Wright-Patterson Air Force Base, Ohio 45433 under a Small Business Research Contract, F33615-90-C-0011, Work Unit 3005-L2-03.

The DKMS project was created to investigate some of the problems that have traditionally restricted the design process. These problems, which include knowledge representation, constraint propagation, model design, and information integration, are addressed by DKMS.

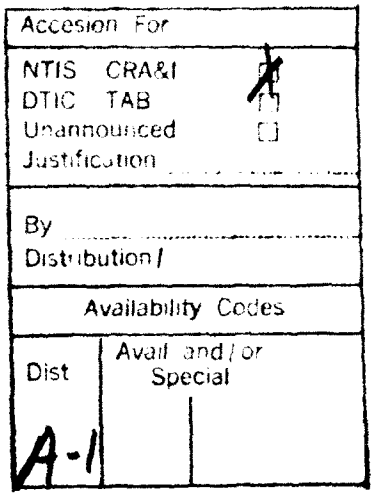
The document is divided into eight sections and an appendix.

1. Introduction
2. Integration Platform
3. Shape-Based Design Knowledge Representation and Reasoning
4. High Productivity Comp.ıter-Aided Design
5. Container Object System
6. Model Design Support Environment
7. Summary
8. References

Appendix

The introduction is an overview of the DKMS effort. Sections 2 through 6 contain beta test feedback for the five subsystems of the DKMS. The summary describes potential areas in

which DKMS could be effectively used. An overview of the major DKMS components is given in the appendix.



1.0 Introduction

1.1 DKMS Overview

DKMS provides a software environment for the development and delivery of intelligent assistants. These assistants can be used for computer-aided design (CAD), computer-aided engineering (CAE), and computer-aided manufacturing (CAM) applications in product design, engineering, manufacturing, and logistics planning.

DKMS can be thought of as an integrated concurrent engineering (CE) system with an environment that includes facilities for 1) design knowledge representation, including shape-based associative retrieval and container objects; 2) intelligent user interfaces, including form-feature interfaces and a generalized constructive solid geometry (GCSG) engine; 3) engineering performance modeling functionality, including constraint management and bond graph techniques; 4) data integration support; and 5) configuration management. This environment is intended to support the rapid experimentation, prototyping, and development of a new generation of integrated engineering and manufacturing decision support applications to meet CE challenges. Most importantly, this architecture will enable the capture and delivery to the engineer of product life-cycle experience relative to manufacturability, reliability, and maintainability (MR&M).

Notably, in the final stages of Phase II, it was realized that project expectations, results, and developed technologies greatly exceeded what was first expected. Not only were the required deliverables completed on time and within budget, but additional materials were produced that contribute to the overall DKMS project. These accomplishments include the following:

1. Completed the detailed design documents, including software requirements specifications documents and user manuals for DKMS and all its subsystems.

2. Manufactured, engineered, and unit tested the major DKMS components.

3. Gathered beta test information from as many sources as possible using as many collection methods as possible to maximize potential benefits.

4. Integrated some key subsystems and technologies into other environments and projects to further their utilization and continue their development and refinement.

5. Developed a marketing direction and interest group for some key DKMS subsystems and technologies.

There were also many potential pitfalls and barriers which have been managed during the Phase II development, including the following:

1. Maintained compatibility with the evolving standards.

2. Maintained in-depth knowledge of government and private initiatives in integration schemes and mechanisms.

3. Addressed the difficulty with and apparent apprehension of integrating and testing systems in-situ with "legacy" production systems.

4. Addressed the relative newness and acceptance of CE concepts.

5. Addressed the enterprise-wide cooperational effort--that would require inter- and intra-company-wide planning and implementation (a multi-year effort)--necessary for DKMS.

6. Addressed the issue of generic design. While design is "generic," each company, site, or department has different cultures, philosophies, methodologies, and tools which make a truly directed and meaningful beta test virtually impossible within the given time frame.

7. Addressed the training effort necessary for DKMS. Some DKMS technologies are quite diverse and forward-thinking; therefore, evaluation tasks typically require an extensive training effort.

1.2 Purpose of this DKMS Document

This document reports the progress of the DKMS beta testing and feedback loop. It also reflects user feedback gathered through surveys (see Appendix), beta testing, demonstrations, technical meetings and presentations, and trade show contacts and follow-ups. Each section is based on a major DKMS component and includes an overview that was the general basis for evaluations that were common between participants. The

overviews provide a higher-level perspective from a systems point of view. Also within each section is a subsection detailing feedback. These subsections identify the requirements necessary to allow technologies from each of the five thrusts of the full DKMS to be successfully integrated into a business organization. The views and comments from many participants are from the standpoint of their own business organizations. Individual viewpoints were also used to evaluate the pros and cons of each thrust. The feedback is then presented as desired extensions and criticisms of the DKMS components.

2.0 Integration Platform (IP)

2.1 IP Overview

The DKMS approach to solving the serious problems and challenges facing U.S. industries is to provide support services that address the assimilation, application, access, and maintenance of the life-cycle engineering knowledge base. This section provides some observations on the complexities of the knowledge flow that DKMS (in particular, the IP component) was designed to facilitate.

The complexity of the life-cycle tasks of maintainable, reliable, safe, high-performance systems that employ the current technology within tight cost and schedule constraints requires the access, assembly, and effective delivery of life-cycle engineering knowledge assistance and information to the design/engineering activities. It does not matter whether these engineering activities occur in product, manufacturing, or maintenance arenas and, as a minimum, such assistance includes the following:

1. Application of previously acquired knowledge in the engineering area to the task at hand. The product designer must have efficient access to the product design knowledge accumulated by his/her predecessors. The manufacturing or process engineer must have access to the knowledge of previous experts.

2. Application of knowledge from outside the engineering area to the task at hand. The product designer must have assistance in the application of the experiences and lessons learned by the manufacturing and maintenance engineering discipline.

3. Access to a trace of the decision rationale that led to the most current state of the product or process definition. The product engineer must be able to quickly uncover the rationale of the product designer. The manufacturing engineer must be able to lay open the decision rationale and assumptions of both the product designers and engineers.

4. Access to the most current state of the product or process configuration description.

5. Man-machine interfaces and protocols which support the capture of product definition data as the decisions relating to that data are made --

> not after the fact. If these don't exist, delivery of CE support is almost impossible. Similarly, the accumulation of the rationale, experience, and knowledge bases that CE support would be based on cannot be affordably achieved without such interfaces.

IP provides the required support for organizing and maintaining standard interface definitions that allow access to the appropriate knowledge as well as routing the integration service requests.

2.2 IP Beta Test Feedback

At the end of Phase II of the DKMS effort, IP was at the beta test stage. The original assumptions regarding the use of the BBN CRONUS software as an underlying support layer proved to be undesirable; a large effort is required to rethink and retool this area. The Integrated Services Planner is a prime candidate for inefficiency due to the inherent complexities of planning algorithms. The IP planning aspect needs more work to design a more efficient or, possibly, an optimized planning strategy. Additional effort will be required to harden the code so that the system will operate reliably in a production environment.

The initial concern of users is that a prototype system must undergo a formal quality assurance program. While this is a standard operating procedure in the software industry when moving from final beta test to production, it was not possible to perform a formal total quality assurance test due to the time constraints of the project.

Another area that must be addressed is system failure recovery. Since IP operates in a distributed environment, many types of problems related to network operation and computer resource availability can be encountered. If a certain machine is down or unavailable or the network is not operating properly, IP must have the facilities to recover gracefully. It is difficult to predict all situations that might arise to cause system failure; therefore, development of failure recovery facilities must address these problems as they are discovered.

Initial IP configurations are not sufficient to support production environments. Because CAD and CAE engineering systems operate on many different platforms, it is critical that IP be expanded to support additional hardware platforms. Since most UNIX systems are

fairly compatible, expansion along the lines of a selected UNIX implementation seems most practical, but it is likely that considerable user interface and network development work will be required regardless of the platforms to which the system is ported. After expansion to other UNIX-based platforms, there is also a need to support low-cost machines such as personal computers (PCs) using either MS-DOS or OS/2 or perhaps DEC VMS-based machines.

For IP to be successful in a production environment, entirely new functionalities are required. Intertool communication, which would allow tools to pass messages to one another without using the Integrated Services Manager/Planner as an intermediary, is necessary. This functionality is particularly useful for tools designed for use in different areas of the same domain (i.e., a CAD tool and engineering analysis tool in the manufacturing domain). Another requirement is higher-level integration support. Currently the IP Data Services Manager operates strictly at the file level; thus, a finer-grained data object management scheme is necessary. This level of object management support could be supplied by the Container Object System (COS) (see Section 5.0), but affected applications would need to interface to the COS directly. The management of data from different applications at the data-element level needs more research and development. The result would be an even higher level of integration and reliability by offering a standard access method for common data. It is believed that this scheme would also be more efficient.

3.0 Shape-Based Design Knowledge Representation and Reasoning (SBDKRR)

3.1 SBDKRR Overview

The physical world is a rich source of constraints for the designer--constraints he/she often attempts to overcome, yet upon which he/she must first rely in order to partition and control design evolutic . Different design disciplines treat shape and form differently. To the architect, shape is a means to express themes. To the microelectronic device designer characteristics of the physical shape (such as the size of a die or intercomponent spacing) are naturally provided constraints that must be managed. To the mechanical system designer, shape is often synonymous with function. Our contention is that much of a designer's rationale is based on shape and form-related considerations; however, materials, processes, and environmental constraints are also important considerations. A primary goal of the design activity is to produce specifications; a major aspect of these is the specification of shape and form. During the SBDKRR development effort, the role of shape-based reasoning was analyzed. Critical relationships were dicovered between shape concepting and both problem partitioning and solution structuring. When a person describes a natural shape, one characterization of the involved cognitive processes consists of the following four generic activities applied recursively:

1. Partitioning – breaking the object into separable elements.
2. Classification – categorizing an element with a prototype.
3. Deformation – changing local/global topology to a prototype.
4. Arrangement – describing relationships between the elements.

The term "recursive application" implies that the same process of shape description is normally applied to each element of the partitioning. This recursion appears to continue until the prototype classifications of the resulting elements can be clearly established. When a person forms concepts of shapes, many of the same cognitive activities are

involved, only slightly rearranged. In the design situation, the shape-concepting process appears to proceed recursively with the following steps:

1. Use "function-to-prototype" knowledge to establish a general shape.
2. Use partitioning to separate elements for further specification.
3. Use functional relations to suggest arrangements.
4. Use deformations to accommodate constraints of physical form.
5. Use arrangements to build the whole from the parts.
6. Use shape description to generate the resulting specifications.

Discovery of this process organization led to some important insights relative to the capabilities required of the form-feature CAD interface or SBDKRR component of DKMS. For example, traditional constructive solid geometry (CSG) systems operate on the assumption that the elemental pieces will be defined first, then composed into the whole. In fact, it appears that this strategy for user interaction only works if an image of the object is in front of the user and he/she can develop a partitioning strategy that can be accommodated by the CSG system. Since current CAD systems were designed to replace or augment the draftsman function (which is merely creating the artifact specification), this style of interface is marginally acceptable. However, to deliver CE knowledge support to the designer, one must develop a geometry-concepting environment that does not assume decisions have been made before it is engaged. We believe we have captured the essence of an interface and a geometry engine that will provide such design-concepting support.

We also believe that the close parallelism between the cognitive activities of design and those of shape concepting and description argue for a central focus on a powerful shape-concepting and manipulation DKMS component. Rather than treating the geometry generation element merely as necessary for artifact specification support, we stress that it is a critical component in the capture of design knowledge, delivery of design advice, and support for the human-design cognition process.

3.2 SBDKRR Beta Test Feedback

It has become clear that one cannot represent and reason about shape without being able to represent meaning. The meaning of shape is the existence of constraints relating uniformities between different life-cycle situations in which shape is significant. Interpretation of shape is distinct from its meaning and depends on the agent-selection scheme and awareness of situations and constraints. We may say that SLOT means slot or POCKET means pocket for part features, but the interpretation of slot in the part may differ depending on our awareness of relations to other situations (i.e., whether our frame of reference is that of manufacturing processes such as milling, grinding, or slotting; or analysis processes such as finite element analysis, mass properties, or kinematics).

Design and development of a situation-based reasoner as an extension of SBDKRR, tailored to the needs of shape-based representation and reasoning, should be developed. In a parallel task, useful input could be gained by gathering and formalizing constraints that describe relations between life-cycle situations of parts. The design could be based on the work of Barwise and Perry in *Situations and Attitudes* [Barwise 83] and the most recent work by Devlin in *Logic and Information* [Devlin 91]. The design and implementation of a prototype situation-based reasoner should also draw on the experience of team members in the Knowledge-Based Systems Lab at Texas A&M. Much of the work done in the DKMS Phase II would allow rapid design and development using the DKMS constraint propagator and SBDKRR reasoning system. The situation system should be designed and implemented not only as a part of DKMS but also with the goal of integration into a CAD and data visualization framework.

To effectively communicate our knowledge of the constraints relating the situations which share uniformities of shape, it is imperative that an effective means of communication be used. It is equally important in the manipulation by and visual presentation to the user of the knowledge stored in the system, that the system use a model or perspective of reality that is compatible with the user's frame of reference and awareness of situations and constraints. Thus, if the user is in a feature-based design mode, the system would use a selection scheme that would present the parts in terms of the feature objects and allow the user to manipulate them in terms of feature parameterizations. Therefore, in terms of display, the features would be enumerated as objects and, in terms of manipulation, they might be represented as situations. For a designer working in terms of a polyhedral

representation, the system would be able to enumerate vertices, faces, and polyhedrons, and allow the user to manipulate these entities directly, even though they may be the result of an approximation of a parametric surface in some other situation. For the user working in terms of parametric surfaces, the part would be represented in terms of its surfaces and the user would be able to manipulate those surfaces using the surface knots. The user may select other situations and perhaps constraints to which he/she would like the system to be attuned. For example, if one is designing a part and would like to be kept informed as to the implications of that part in terms of manufacturability within a certain constrained situation (such as <machine bridgeport>), it should be possible to attune the system to that situation and its background constraints and, if necessary, add to or remove constraints.

In our endeavor to represent and reason about the life-cycle implications of shape, and indeed the meaning and interpretation of shape, it is necessary to gather case studies from industry. These case studies will be a sampling of constraints that exist between different life-cycle situations. Encoding these constraints and situations in the framework of situation theory will provide useful input into the design and development of the situation-based reasoner. The situations envisioned will be drawn from concept design, costing, detailed design, manufacturing, process planning, and maintenance for a selection of parts. These case studies could also be used as a test suite for other research efforts. It is important that the case studies be sanctioned by independent groups such as the National Institute of Standards and Technology (NIST) or Society of Manufacturing Engineers (SME) as a test suite or benchmark for Shape-Based Reasoning and Representation Systems so that systems can be independently validated.

In general, current CAD systems can be classified as semantically impoverished. It is not possible to represent the meaning of shape in a CAD system because its representation is primarily focused on topology and is optimized for speed of geometric and topological operations. By integrating High Productivity Computer-Aided Design (HPCAD) system tools and the situation-based reasoner, one can take advantage of the ability of the HPCAD system to rapidly present and manipulate data visually, and the capability of the situation-based reasoner to deal with meaning. This would allow one to describe life-cycle situations and define constraints between them directly on visual presentations of these situations. For example, a situation may be described in which a part has a slot and a manufacturing situation may be described which contains certain equipment. Then constraints can be defined relating the capabilities of the manufacturing situation to the part feature. The

marriage of CAD and situation-based reasoning will make it possible for the designer to assess the life-cycle implications of design decisions via the constraints relating the different aspects of the product life cycle. Modularity and user interface protocols should be used to allow interchange or customization of reasoners between CAD systems.

Separation of the man/model interface from the CAD system and the situation-based reasoner is an important design decision because it allows one to more easily include other means of representation and reasoning, other situation-based reasoners, and other CAD systems.

4.0 High Productivity Computer-Aided Design (HPCAD)

4.1 HPCAD Overview

Construction of an HPCAD interface is crucial to the DKMS project and several major components to enable the acquisition of design knowledge that is form- and shape-indexed. Existing CAD systems are constructed on a single geometric representation. This limits the primitives that the user has to work from and also restricts the availability of analysis programs. The thrust of the HPCAD Toolkit in this area was to develop the mathematical foundations and algorithms for a generalized constructive solid geometry (GCSG) system that will serve as a key component in the advanced DKMS. In the multirepresentation environments expected in engineering/manufacturing systems, GCSG capability forms the basis for such high-productivity interfaces, because it makes possible the "seamless" (and rapid) manipulation of multiple CSG representations. The "seamless" nature of the manipulation allows the system to translate between various representations without user intervention. A majority of the work in the GCSG area is focused on: 1) the implementation of efficient algorithms for performing Boolean operations over heterogeneous representations for solid objects, and 2) the implementation of matching algorithms for these objects. The matching algorithms will be needed to perform comparisons among objects, either to find a similar object (knowledge delivery) or identify differences (knowledge acquisition).

HPCAD will be responsible for translating geometry from one representation technique or parameterization to another and from one data representation to another. HPCAD bridges these representational discontinuities by providing a convenient means of creating services for translations and for performing Boolean operations on surfaces created in different modeling systems from different parameterizations. The HPCAD Toolkit also facilitates the creation of customized CAD systems or can be used in a supporting role, such as in a shape-based reasoning application like SBDKRR, where the Toolkit routines may be used as predicates.

The Toolkit will support the viewing and manipulating of Bézier and Non Uniform Rational B-Spline (NURBS) surfaces, solids via polygonal boundary representations (both winged-triangle and winged-edge representations) and CSG, and hybrid surface/solid representations using GCSG. Solids may be modified using Euler operators and CSG. The Toolkit will support the algebraic (exact) translation between parametric as well as approximate translations.

4.2 HPCAD Beta Test Feedback

HPCAD needs full compliance with a number of standards and other nonstandard file formats (including Product Data Exchange using Step/Standard for the Exchange of Product Data (PDES/STEP), Initial Graphics Exchange Specifications (IGES), Data Exchange Format (DXF), etc.) to be fully accepted as a viable toolkit in a production environment. HPCAD lacks the capability to store created objects or designs and share common data. HPCAD should have an interface protocol to the COS system to insulate itself from platform dependencies.

For HPCAD to be more generic, it must have more common interfaces available for development (e.g., X Windows/Motif, OS/2 Presentation Manager, etc.). The HPCAD Toolkit needs an intelligent user interface that is responsible for how "objects" are "presented" to the user screen and how these "presentations" are interpreted in varying contexts.

The Design History Manager cannot be fully realized without using a specific user interface on a specific platform. Currently, all DKMS work is performed as modularly and generically as possible; this limits any specific customizations.

The HPCAD Toolkit provides the low-level building blocks (such as vector utilities) and some higher-level construction utilities. While it is certainly possible to build CAE applications from the Toolkit, the Toolkit may be too primitive for the effective creation of these applications. There is a need to provide higher-level functions that are common to CAE applications, such as convex hull generation, surface triangulation, convex decomposition, and other building blocks that do not require the programmer to "reinvent the wheel."

5.0 Container Object System (COS)

5.1 COS Overview

COS proposes a new knowledge representation for CE design called *container objects.* These objects incorporate the best qualities of earlier representation efforts. Container objects have been demonstrated to be an effective solution to the problem of representing design knowledge within CE applications. COS is used by the other DKMS components as a common representation for design knowledge. It is composed of: Perspective Component, Container Object Component, Constraints Component, Generic Functions and Method Component, and Persistent Storage Component. COS borrows heavily from earlier *object-oriented* representation schemes and from later derivatives called *composite-object systems.*

COS extends the notion of objects to make them recursive under composition, thus enabling instantiation of a group of objects as an entity. This is useful when relative relationships between members of the group must be isomorphic for distinct instances. A composite is defined by a template that describes the subobjects and their connections. These subobjects are created by an instantiation process and are describable in a class inheritance network. One benefit of composite object classes is the ability to make modified versions of a template by creating a new subclass which inherits the properties of the superclass. Facilities for making composite objects are not common in object-oriented languages, but are fairly common in application languages, such as those for describing circuits and layout of computer hardware.

Composite objects have the following features to support DKMS:

1. Composite objects are specified by a class containing a description that indicates the classes and interconnections of the parts. The use of a class makes instantiation uniform.

2. Instantiation creates instances corresponding to all the parts in the description. The instantiation process must track correspondence between the parts in the instantiated object. It fills in all the

connections between objects and must permit multiple distinct uses of identical parts.

3. The instantiation process must be recursive to allow the use of composite objects as parts. For programming convenience, the system must either flag as an error the situation where a description specifies using a new instance of itself as a part or support "lazy instantiation." A description of a part which includes itself can result in the instantiation of objects with an unbound size. Alternatively, instantiating subparts on demand (lazy instantiation) would allow the use of potentially unbounded objects because the subparts are generated only on use.

4. It must be possible to specialize a description by adding new parts or substituting for existing parts. The description language must allow specialization of composite objects with a granularity of changes at the part level.

5.2 COS Beta Test Feedback

The prototype COS is a vital representation component for all other DKMS elements. While it is very powerful and innovative as a knowledge representation system, it is implemented in Common Lisp. Although Lisp is a very powerful implementation language and affords good portability to other platforms, the availability of Lisp compilers cannot be guaranteed for all platforms. Hence, this restriction must be removed by writing a version of COS in a more prevalent language. The suggested target language is C++. The conversion effort should be language-independent while defining the basic data structures used for representing container object information. This language independence will make COS independent of any underlying object base that is chosen to handle persistence. Since Lisp and C++ both provide powerful representations for data, this will not be a trivial task. Lisp and C++ also have an enormous number of differences in their approach to representing programs; therefore, a straight translation of concepts will generally result in an inefficient and unmaintainable program. Consequently, the algorithmic elements of COS which take advantage of the capabilities of Lisp will have to be identified, and an approach more suitable to C++ will have to be used to translate them. The largest task of converting to C++ will be aimed at the interface scheme for use in its environment. This task promises to be one of the greatest challenges of the entire translation effort. The difficulty involved here is that the effects of commands in a Lisp environment tend to be persistent; that is, the effects of one command in the Lisp environment are present at the

invocation of the next. This greatly simplifies interface issues in a Lisp environment, but not in a typical C++ environment. Furthermore, use of assessor functions (those functions designed to retrieve data from data structures) is well-supported by Lisp, whereas C++ uses no assessor functions. Thus, an alternate scheme must be developed for the converted COS.

The current COS prototype is susceptible to unforeseen problems and user-caused errors. The error-handling capabilities must be expanded to gracefully handle errors and user/programmer mistakes. Furthermore, the prototype satisfies the DKMS Phase II representational requirements as a proof of concept. While it is currently efficient enough for most applications, it remains to be seen how well COS scales up to large, memory-intensive applications. The efficiency aspects of COS must be investigated to guarantee that performance is acceptable over a wide range of applications.

Another area that requires more investigation is the use of COS in a distributed environment such that it encourages concurrency in design and manufacturing processes. While it will be perfectly suitable for single-user applications as it stands, COS should provide some utilities for supporting distributed applications. Currently, COS relies on the underlying transport layer of the persistent object for this capability; it should be added at a higher level to explicitly allow the COS user more control over concurrency when needed. For example, two design teams working on a new product design wish to work on separate components of the product independently. Since component testing will frequently require accessing the entire product, both teams will import copies of the product representation to their local machines. However, this situation can easily lead to read/write and write/write conflicts, where one group unwittingly destroys the changes another group has made by simply writing out their version of the product immediately after the second group has saved theirs. To prevent this, the following scenario is envisioned. Instead of importing the entire representation of the product with read and write privileges, each group would be able to use write privileges on their respective components only, but still be able to read the representation of the entire product.

6.0 Model Design Support Environment (MDSE)

6.1 MDSE Overview

A primary component in any design support or automated "designer" system is the engineering analytic model(s). One goal of DKMS is to support the development/refinement of the models of the physical systems or processes used in that design process

A large part of engineering design involves manipulating models; yet, most computer models today are coded such that they are resistant to change. Any changes to the existing models frequently require extensive recoding of very old and largely undocumented code. New model developments are equally difficult because the engineer has no means of expressing the model concepts to a computer in a way natural to him/her. What is needed is an intelligent model-building system that knows about model concepts, solution technique types, and implementation methods. The resulting modeling support environment will provide capabilities in the following areas:

1. Model rationale and knowledge-base capture.
2. Constraint management for supporting the derivation of complex systems models from first principles as well as test data results.

Of these, the most powerful is constraint management. Design is constraint-oriented; much of the design process involves the recognition, formulation, and satisfaction of constraints. Constraint management can aid designers in identifying and exploring the boundaries of the design space to determine the most important design parameters, specifications, and constraints, and in evaluating the global performance of the alternatives to select the most appropriate ones for detailed analysis and refinement.

The main functionalities to be provided by MDSE include the following:

1. Causality and dependency determination via bipartite matching.
2. Strong component (simultaneous constraints) detection.

3. Consistency verification.

4. Solution sequence generation.

6.2 MDSE Beta Test Feedback

To simplify MDSE delivery and enable integration of a commercially supported Computer Algebra System (CAS), MDSE must be rehosted on a commonly available workstation. MDSE is currently implemented on a Symbolics Lisp machine, which has provided excellent support for the initial development. While beneficial in initial development, the Symbolics machines have limited distribution. This limited distribution makes delivery of the code difficult; lack of user familiarity with these machine makes training costly. In addition, the only CAS available for Symbolics is Macsyma. Macsyma does not support a programmer interface; therefore, any integration of Macsyma into MDSE can only be accomplished at a superficial level and may require as much effort as rehosting MDSE on a workstation. A workstation, in this context, can include any stand-alone, single-user computer with a graphical user interface. This includes such machines as the Sun SparcStation but could also include large 486-based PCs with MS-DOS 5.0 running MicroSoft Windows. The choice of workstation would be dictated by the availability of a CAS for that platform.

A key feature of MDSE is its ability to generate multiple solution sequences for a given model. These solution sequences specify which equations are to be solved for what variable, the order in which they should be solved, and which sets of equations must be solved simultaneously. It is possible to improve the efficiency of the code that is produced by that system by examining multiple solution sequences. Generating all possible solution sequences is, worst case, an intractable problem. In this worst-case scenario, the possible number of solution sequences grows combinatorically with respect to number of inputs. Clearly, this is unacceptable. To remedy the problem, the algorithm has been modified to return when a specific, predetermined number of solution sequences are found. While this solves the space/time problem by limiting the amount of computation and storage needed, it does so in a manner that might ignore very desirable solutions. A more sophisticated technique for searching the solution space may enable MDSE to find an optimal sequence.

The ability to encapsulate information in a subprogram has proven to be useful in the development of software systems. The need to break down a problem and write programs to solve each subproblem is a capability that all mature programming techniques contain. This holds true for MDSE as well, yet developing this capability requires extensions to the constraint management theory that is at the heart of MDSE. Constraint management is a collection of graph search techniques which allow a program to determine the variable for which each constraint must be solved and the order in which the constraints must be solved. This technique is used to generate the solution sequences in the current MDSE. The original work in constraint management required that each constraint be solvable for any number of variables. A subprogram, however, has a specific list of inputs and outputs for which it can be solved. Since a subprogram can only be solved for certain variables, it violates one of the precepts of constraint management, therefore invalidating the technique for use with them. This incompatibility means that the theory behind constraint management must be extended to cover this new case if subprograms are to be included in MDSE. Extending constraint management in this manner would allow programs within MDSE to include previously defined programs in their definition. A useful side effect of this extension is the ability to include user-defined functions and subroutines in MDSE. This would allow a user to specify a subprogram externally and reference it in the definition of an MDSE program. Additionally, these "known" solutions or subprograms could be built as packaged libraries, thereby enhancing efficiency tremendously.

MDSE uses a CAS to solve equations for a particular variable. Ideally, a CAS would always return a closed-form solution for the expression. Unfortunately, in many cases this is not possible and numerical methods must be used. The closed-form solution is always the most desirable because it is completely accurate, can be solved quickly, does not rely on an initial "guess," and cannot fail to converge. Clearly, the more frequently a closed-form solution can be found, the more robust MDSE would be. The CAS currently embedded in MDSE was created as an experimental tool and is capable of finding closed-form solutions in a very restricted set of equations. While it is useful in establishing the viability of the concepts behind MDSE, it is not sufficient to solve the kind of problems MDSE was designed to handle. Further work needs to be done to extend MDSE to solve larger, less-restricted sets of equations.

Communication protocol for CASs should be defined to enable full modularity between systems; this would enable MDSE to use different CASs by rewriting the protocol handler

for a given CAS. This protocol must delineate how equations are passed to the CAS for solution and how results of that solution procedure will be passed back to MDSE. The exact definition of the protocol will depend on the specific CAS chosen.

To simplify the initial construction of models, MDSE should be extended to include the ability to define models based on an icon-driven interface. The icons would represent subcomponents of a more complex device, such as the piston, cylinder, and orifice of a pump. These subcomponents (or macros) would contain information concerning different features of the object. For example, these macros could be assigned a "material type" that could be combined with others to represent a higher-level macro concerning its physical properties. In addition, macros could contain a set of relationships commonly used in modeling. These simple macros would serve as a shorthand for collecting information on a system. These macro interactions could be described during the model specification stage within MDSE.

The majority of engineering constraints are not expressed as equations but as inequalities. Since good design is an optimization of a number of acceptable alternatives, its requirements are likely to be phrased in terms of what qualifications an acceptable design must meet. The engineer is expected to find an optimum design which meets these requirements. Directly supporting this type of constraint will enable a modeler to express his/her model in terms that correspond directly to the specification it is intended to fulfill; by extending MDSE to handle inequalities, it can produce a model that finds an optimal design. This changes the model from simply predicting system performance to actually suggesting the design parameters--a clear improvement for users of this technology.

7.0 Summary

The current DKMS could be usable as a base or starting point for a CE platform for almost any engineering automation effort in the Government, defense contractor, and commercial industrial sectors. DKMS could provide a quantum improvement over any available design automation concept available today. It could also serve as a means to promote better university/industry/government ties because it would provide a direct vehicle for moving design, manufacturing, and field experience into the classroom. Initial production installations are anticipated to very aggressive small business communities that must leverage scarce resources, 2) Department of Defense (DoD) installations for which management of knowledge bases over long life-cycle weapon systems is a critical issue, 3) National Aeronautics and Space Administration (NASA) Space Station and deep space missions for which knowledge bases will span multicareers, and 4) commercial industries for which reduction in product development time is critical to the maintenance of a competitive position. The results of the DKMS project have been focused primarily on mechanical device design support. By including industrial, university, and government feedback as a part of the Phase II development, Knowledge Based Systems, Incorporated (KBSI) is in a very good position to set up either investment or industry development partnerships to fund the Phase III commercialization that could address other domains as well. Because of the modular nature of the approach taken, KBSI will also be able to structure investment support for commercialization of DKMS one piece at a time. This modularity, combined with the diverse body of feedback from the potential user community and beta test sites, will also allow KBSI to fund selected pieces of this commercialization itself.

The technology resulting from the DKMS project will provide support for design engineers to better integrate the trade-off of various design attributes such as performance, cost, schedule manufacturability, and supportability. It will also significantly reduce the engineering cycle time and manpower requirements for both initial product design and sustaining engineering of a product. The resulting system could truly provide a framework for CE initiatives beyond those demonstrated in this Phase II effort. By supporting the knowledge-based delivery of production and maintenance experience to the initial product

designers, an order of magnitude reduction in redesign and engineering change requests will be realized. Finally, because of the integration support and HPCAD support offered by DKMS, it will find use in many areas other than product design. For example, in manufacturing planning and production planning, access to the design rationale and design knowledge bases will allow automation of a number of these "manufacturing engineering" activities and will significantly reduce quality and reliability problems in the final product.

8.0 Bibliography

[Barwise 83] Barwise, J., and Perry, J. *Situations and Attitudes*. MIT Press, Cambridge, MA, 1983.

[Devlin 91] Devlin, K., *Logic and Information, Volume I: Situation Theory*. Cambridge University Press, Cambridge, MA, 1991.

[DKMS 90] *A Design Knowledge Management System (DKMS)*, SBIR Phase I Final Report, Knowledge Based Systems, Inc., Dayton, OH, April, 1990. Contract F41622-89-C-1018, AFHRL, WPAFB, OH.

[KBSI 91a] Knowledge Based Systems, Inc. *Beta-Test Training Document*, KBSI-DKMS-90-STR-01-0492-01, Volume 1, AL/HRGA, Wright-Patterson AFB, June, 1991.

[KBSI 91b] Knowledge Based Systems, Inc. *Software Design Document for the Container Object System*, KBSI-DKMS-90-SDD-04-0392-04, Volume 4 of 5, AL/HRGA, Wright-Patterson AFB, October, 1991.

[KBSI 91c] Knowledge Based Systems, Inc. *Software Design Document for the High Productivity CAD Toolkit*, KBSI-DKMS-90-SDD-03-0392-04, Volume 3 of 5, AL/HRGA, Wright-Patterson AFB, October, 1991.

[KBSI 91d] Knowledge Based Systems, Inc. *Software Design Document for the Model Design Support Environment*, KBSI-DKMS-90-SDD-05-0392-04, Volume 5 of 5, AL/HRGA, Wright-Patterson AFB, October, 1991.

[KBSI 91e] Knowledge Based Systems, Inc. *Software Design Document for the Integration Platform*, KBSI-DKMS-90-SDD-01-0392-04, Volume 1 of 5, AL/HRGA, Wright-Patterson AFB, October, 1991.

[KBSI 91f] Knowledge Based Systems, Inc. *Software Design Document for the Shape-based Design Knowledge Representation and Reasoning System*, KBSI-DKMS-90-TR-02-0492-01, Volume 2 of 5, AL/HRGA, Wright-Patterson AFB, October, 1991.

[KBSI 91g] Knowledge Based Systems, Inc. *Software Requirements Specification for the High Productivity CAD Toolkit*, KBSI-DKMS-90-SRS-03-0392-01, Volume 3 of 5, AL/HRGA, Wright-Patterson AFB, July, 1991.

[KBSI 91h] Knowledge Based Systems, Inc. *Software Requirements Specification for the Container Object System*, KBSI-DKMS-90-SRS-04-0392-01, Volume 4 of 5, AL/HRGA, Wright-Patterson AFB, July, 1991.

[KBSI 91i] Knowledge Based Systems, Inc. *Software Requirements Specification for the Model Design Support Environment*, KBSI-DKMS-90-SRS-05-0392-02, Volume 5 of 5, AL/HRGA, Wright-Patterson AFB, July, 1991.

[KBSI 91j] Knowledge Based Systems, Inc. *Software Requirements Specification for the Integration Platform*, KBSI-DKMS-90-SRS-01-0392-01, Volume 1 of 5, AL/HRGA, Wright-Patterson AFB, July, 1991.

[KBSI 91k] Knowledge Based Systems, Inc. *Software Requirements Specification for the Shape-based Design Knowledge Representation and Reasoning System*, KBSI-DKMS-90-SRS-02-0492-01, Volume 2 of 5, AL/HRGA, Wright-Patterson AFB, July, 1991.

[KBSI 91l] Knowledge Based Systems, Inc. *Container Object System (COS) Programmer's Manual*, KBSI-DKMS-90-SUM-04-0392-01, Volume 4 of 5, AL/HRGA, Wright-Patterson AFB, May, 1991.

[KBSI 91m] Knowledge Based Systems, Inc. *High Productivity CAD (HPCAD) Programmer's Manual*, KBSI-DKMS-90-SUM-03-0392-01, Volume 3 of 5, AL/HRGA, Wright-Patterson AFB, May, 1991.

[KBSI 91n] Knowledge Based Systems, Inc. *Integration Platform (IP) User's Manual*, KBSI-DKMS-90-SUM-01-0392-01, Volume 1 of 5, AL/HRGA, Wright-Patterson AFB, May, 1991.

[KBSI 91o] Knowledge Based Systems, Inc. *Model Design Support Environment (MDSE) User's Manual*, KBSI-DKMS-90-SUM-05-0392-01, Volume 5 of 5, AL/HRGA, Wright-Patterson AFB, May, 1991.

[KBSI 91p] Knowledge Based Systems, Inc. *Shaped-Based Design Knowledge Representation and Reasoning (SBDKRR) User's Manual*, KBSI-DKMS-90-SUM-02-0492-01, Volume 2 of 5, AL/HRGA, Wright-Patterson AFB, May, 1991.

[KBSI 91q] Knowledge Based Systems, Inc. *Technology Impact Report*, KBSI-DKMS-90-STR-01-0292-03, Volume 1 of 1, AL/HRGA, Wright-Patterson AFB, February, 1992.

Appendix: Beta Test Survey Report

Table of Contents

List of Figures

Preface

This document is the foundation for discussions related to establishing potential beta test site users for the Design Knowledge Management System (DKMS) and acquainting them with the current software capabilities. This report is also to be used as a survey for evaluating the applicability of the various DKMS components within the beta test site user engineering environment. This manual begins with a brief introduction to DKMS; each subsequent section addresses a major component of DKMS.

1.0 Design Knowledge Management System (DKMS)

1.1 Introduction

DKMS is a software environment which provides support for and control over the design development process. It is a heterogeneous integration environment designed to be configurable to the design development practices of a particular site.

DKMS was conceived in response to the difficulty of acquiring, representing, reasoning with, and applying design knowledge. With the advent of concurrent engineering (CE) (made possible by more powerful and economical computer hardware resources), the complexity of design knowledge management has increased dramatically. As designs become more complicated, more management is required to ensure that the software systems are produced in a consistent manner, on time, and within budget. It is also important to ensure that the systems are reliable and maintainable. Another design factor is the need to use past experience in the development of new products. This may be as simple as reusing a function produced for another system or as complicated as reusing an entire design knowledge base or development strategy.

This report reflects the functionality, features, key components and their uses, and any pertinent underlying concepts of the various DKMS components as they currently exist.

1.1.1 Purpose of this Document

This Beta Test Survey Report aids in understanding and evaluating the key concepts of DKMS as they relate to the beta-release software. This document is intended as a management overview, not an all-inclusive report of DKMS capabilities.

In this release, the document presents a brief description of each DKMS component and outlines the roles and underlying concepts of each. This report is not meant to be used as a tutorial or supplement to the training guide. Installation procedures and tutorial information

are detailed in the actual Beta Test Training Document [DKMS 91]. This document outlines the content and format the DKMS software will contain in the current and future releases. It also elicits feedback to make future releases of this and other documents easier to use.

1.1.2 Organization of this Document

The remainder of this report is divided into five major sections reflecting the major DKMS components as they currently exist. Though we expect no major changes in the document or software structure, the contents of this and other DKMS manuals will grow as the system reaches completion.

This report is divided into the following sections:

1. Integration Platform (IP).
2. Shape-Based Design Knowledge Representation and Reasoning (SBDKRR).
3. High Productivity Computer-Aided Design (HPCAD).
4. Container Object System (COS).
5. Model Development Support Environment (MDSE).

Each section presents an overview of the functionality of a component and the key concepts as they relate to the overall DKMS strategy.

2.0 Integration Platform (IP)

2.1 Introduction

IP operates in a distributed, heterogeneous environment and attempts to provide a realistic integration strategy that supports function and data integration in addition to providing design artifact management facilities. IP allows the user access to various utilities and tools via services provided in the DKMS environment. It also supports the design process by managing the design artifacts produced during development by preserving their design history.

2.2 Overview of the Functionality

2.2.1 IP as Part of DKMS

IP allows the user access to various utilities and tools via *services* provided in the DKMS environment. A service is an activity/program that is supported by the system and can invoke tools and programs on objects or object sets to produce the desired result or side effect. Using the interface provided by IP, the user can list the services the system supports and add new ones to the database of services. The user can also browse and query the service descriptions.

Figure 1 illustrates the IP architecture. The Platform Interface handles the flow of information and data to and from the applications connected to the IP. The user can directly access IP via the User Session Interface. This application program allows the user to make queries and requests directly to the platform. The Artifact Manager is the component of the system that handles all information stored about design artifacts. Information such as latest version, location, artifact format, and access policies is managed by the Artifact Manager. The Integration Service Manager (ISM) works in conjunction with the Integration Service Planner (ISP) to handle service requests from the users and applications for access to the system. The Facilitator executes the requested services either locally or remotely across the

network. The Network Manager handles service requests and data that are sent over the network. The Data Services are simply the collection of programs and utilities that provide the services. The functions of the various components are described in greater detail in the following sections.

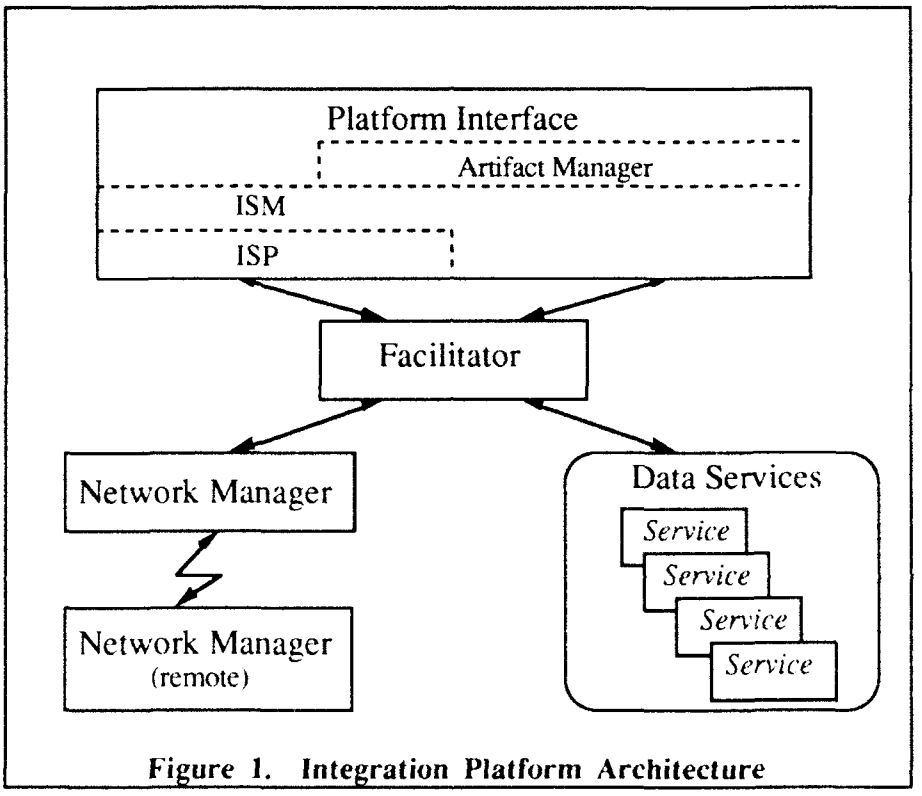


Figure 1. Integration Platform Architecture

2.2.2 Key Components and Their Use

IP is responsible for monitoring and controlling the generation and execution of integration service plans. The key IP components for providing services are the ISM and ISP. The ISM is responsible for receiving and executing integration service requests. By doing this, the ISM spawns background plan management processes for each request, monitors the progress, and collates the results. Part of the process of providing a service is the generation of service plans to provide more complex services; plan generation is performed by the ISP.

Figure 2 shows the key functionality that supports service requests. The user can define new services through the Register Service component. These services are then available to all hosts within the DKMS platform. The ISP can use these service descriptions to build more complex services.

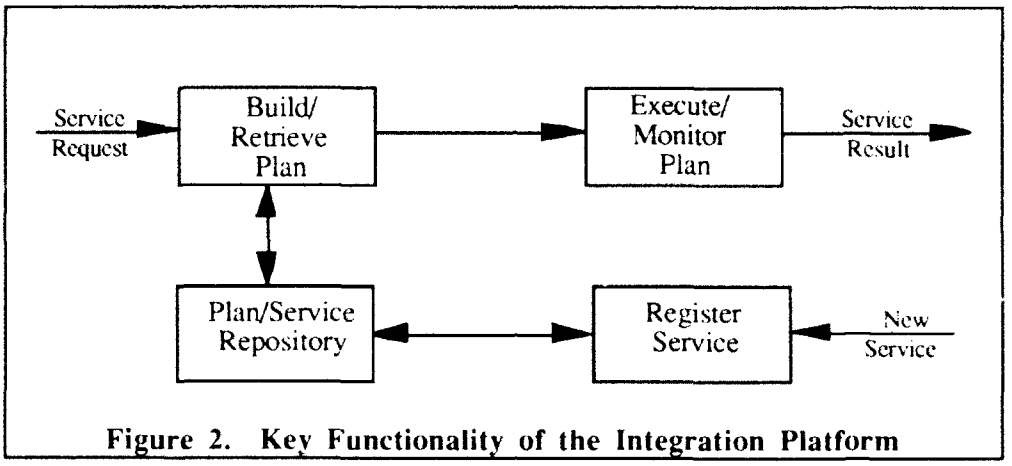


Figure 2. Key Functionality of the Integration Platform

Using the User Session Interface to IP, the user can add new services, browse existing ones, or use information provided in existing service descriptions in their own services. The user can also invoke the ISP to generate service plans that he/she could edit or extract. The extracted plans can be included in the user application code.

The Artifact Manager provides a high-level object and knowledge-based system that has the flexibility to change and the ability to track entities in a software system as it develops. The functionality of the Artifact Manager includes information/data history management, versioning and configuration management, access control, construction of composite objects (across multiple databases), and constraint propagation. The ability to manage this information comes from the COS (see Section 5.0 for more information on COS), which is called by the Artifact Manager to access artifact information.

The key components of the Artifact Manager are:

1. COS – a flexible, high-level object-oriented system.
2. Versioning and Configuration Management System – allows different levels of version control.
3. Access Control System – allows user authorization and enforces the authorization control.

4. Browser/Editor System – allows user to browse a heterogeneous mix of databases and extract the required information.

2.2.3 Underlying Concepts

IP takes a service-based approach to integration. Rather than focusing on the construction of an "integrated system," emphasis is on the "integration services" IP and functional applications provide.

The advantages of this approach are:

1. Flexibility – allows the use of machine-specific services of one host on other machines.
2. Shorter Software Development Time – facilitates reuse of code in the form of services (because the services are designed to be independent modules).
3. Integration – assists in the use of new tools with the DKMS platform.
4. Reuse - existing data, information, and knowledge can be reused.
5. Remote Execution – automatically handles remote execution without user intervention, thus shielding the user from the intricacies of network protocols.
6. Product Data Evolution – includes factors like problem report generation, change request processing, message passing, status report generating, and status browsing.
7. Knowledge Base Evolution Control – includes user-defined factors like rules and organization policies, constraints on the product base, policies that require approval, context, and principles of design.

The purpose of the IP is to realize a new approach to the challenge of integrating a large software system. This is accomplished by assisting the user in providing data in the form required by the various components using the platform. The services approach taken by the IP allows the generation of plans based on requests for data or functionality. To carry out the plans, services are provided which perform a specific data manipulation operation or provide a specific function. These operations can be linked together to create a new kind of service. By providing a networking capability, the system can carry out plans and services on other hosts and, therefore, facilitate integration even further.

3.0 Shape-Based Design Knowledge Representation and Reasoning (SBDKRR)

3.1 Introduction

In SBDKRR, it is believed that "featurization" of a part is an interpretation process that occurs within a particular context. The same physical phenomena (a part) is partitioned and abstracted differently by different agents in different contexts. Different agents in different contexts are attuned to different perceptions. This difference of attunement implies that different information is extracted from the same phenomena. Even in simple natural language phenomena, in which one might suppose that exactly the same attunement to the words in a sentence exists, interpretation of a sentence changes from context to context. Attunement and interpretation are influenced by the acquired knowledge of the agents in the context. The implications of this view of featurization as an interpretation process impact many aspects of CE applications, from the user interface to the information integration mechanisms. For example, if one accepts that expert performance in the complex domains of engineering and manufacturing requires expertise acquired through experience and specialized training, it is unreasonable (and probably undesirable) to expect designers to effectively think in terms of manufacturing features except in the most trivial cases. This belief is based on a wealth of historical evidence arising from the Group Technology domain.

The technical approach to the shape-based representation and reasoning problem may be broken into three main themes: 1) representation, 2) reasoning, and 3) architecture. The knowledge representation scheme is recursive and includes the concepts of frame of reference (FoR), granularity, and constraints. SBDKRR uses a multilayered reasoning approach that implements 1) geometric algorithms for extracting uninterpreted patterns from part descriptions, 2) graph search algorithms for prototype matching of shapes, and 3) Artificial Intelligence (AI) inferencing algorithms for interpreting the prototypical shapes.

3.2 Overview of the Functionality

3.2.1 Shape-Based Reasoning as Part of DKMS

The complexity of today's products requires the assembly of large teams to accomplish the CE of a product throughout its life cycle. At present, these teams are used as the life-cycle design and analysis information pipelines through which the emerging design is iterated until it satisfies the design requirements. The "To Be" situation emphasizes parallelism of tasks that are driven by design ideas constrained by the life-cycle implications of those ideas. Initiatives towards CE must include many different organizations in design, engineering, manufacturing, and field support that operate as a virtual design team. The key to successful CE application is the capture of life-cycle knowledge and the timely, effective delivery of it to the appropriate decision-making event. Much of the knowledge applied in or communicated between these activities cannot be represented in linguistic terms easily because it is primarily shape-based or shape-indexed. Shape-and feature-based knowledge refers to that engineering, manufacturing, and maintenance knowledge associated with the geometric properties of the part. Because these shapes do not necessarily have names, they cannot be represented in traditional knowledge representation schemes. Also, traditional, literal, pattern-matching schemes do not work for purposes of knowledge retrieval or reasoning on these shapes.

The part representation used by conventional computer-aided design (CAD)--including solid modeling--systems is incomplete in terms of the semantic and qualitative part descriptions needed for CE. In addition, the closed architectures of these CAD systems makes delivery of the life-cycle knowledge to the decision-making event difficult. However, for effective CE applications, we would like to identify problem areas in a design and provide recommendations interactively during the design-creation process. This problem of impoverished part representation also has inhibiting effects on the cost-efficient application of computer-aided engineering (CAE). For example, a hole is represented as a cylindrical face connected to the faces around it in a solid modeler. This representation is not suitable for grid refinement algorithms used in finite element analysis that need to know 1) that the hole is a stress raiser and, consequently, needs a finer mesh; 2) that the cylinder represents a hole that may be mapped by a process planning system to drilling, reaming, or

boring operations; or 3) that the hole presents problems to manufacturing because of the way it interacts with other features.

SBDKRR provides the capability to represent antecedents or consequences of design knowledge that contains arbitrary geometric structures (generalized shapes). SBDKRR also addresses the application of these results to feature-based knowledge representation used for geometric references that have named prototypes (e.g., hole, slot, pockets).

3.2.2 Key Components and Their Use

In the development of the shape-based design knowledge representation scheme, some key questions that need to be addressed include:

1. How do we create a representation for form features that allows us to map into different frames of references?

2. Do we need to use a nonmanifold topology to represent our part geometry in order to keep our representation neutral (i.e., when we need manifold topologies, why don't we use a mapping function)?

3. When we reason about shape, are we using an internalized (Naive) representation that is nonmanifold and non-Euclidean?

4. How do we use this representation to deliver knowledge to and from CAD systems?

The representation for shape-based knowledge that has been developed is outlined in this section. Note that by explicitly including the concept of dimensionality in granularity, there is not always a tidy substructure/ superstructure or refinement relation between granularities. This allows one to speak of an Affine, Cartesian, or manifold granularity of a part. An example of the nonmanifold topology of this scheme used to represent volumetric features is shown in Figure 3.

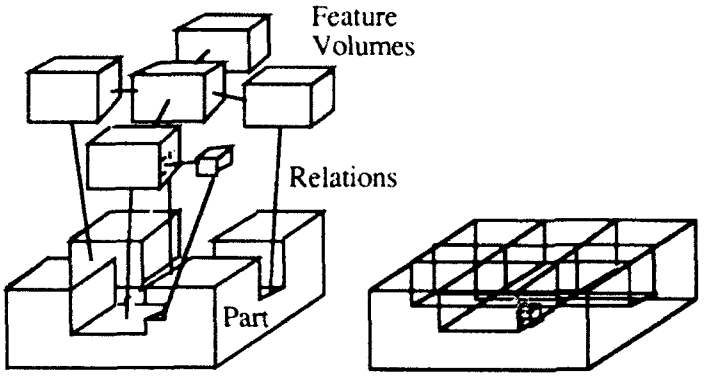


Figure 3a
A part with feature volumes.

Figure 3b
Cellular representation of part and features.

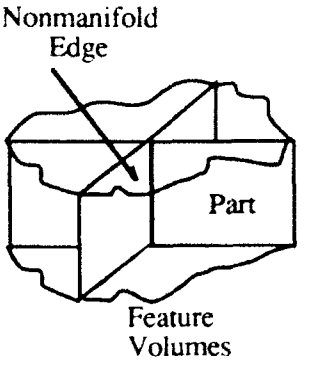


Figure 3c
An edge sharing more than two faces is nonmanifold.

Figure 3. Examples of Nonmanifold Topologies

3.2.3 Key Concepts

3.2.3.1 Form-Feature Representation

A shape-based knowledge representation should include 1) a unique identifier for space description elements, 2) the ability to describe relations about shape compositions, 3) FoR (see Figure 4), and 4) a set of logical constraints describing the shape type or composition. The initial definition for a form-feature representation is:

(Shape N F π C S) where,

N is a feature name,

F is the FoR,

π is a set of primitive element specifications or feature specifications, and

C is a set of constraints (geometric, topological, etc., relating elements of π in FoR ***F***).

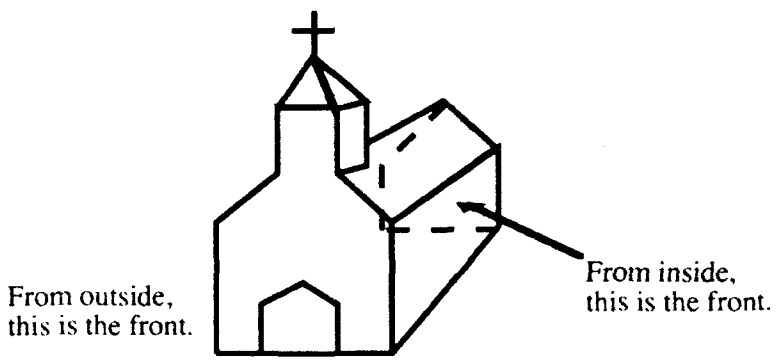


Figure 4. An Example of a Frame of Reference

3.2.3.2 Frame of Reference

It is important to establish an FoR for the spatial descriptions since 1) the spatial prepositions used may be used in a deictic, intrinsic, or extrinsic manner; 2) adjacency relations are sensitive to the FoR granularity; and 3) all spatial metrics must be declared relative to some coordinate frame. To permit reuse of FoRs, the inheritance of reference frames is allowed.

An *FoR* is defined as:

(FoR N G D L S) where

N is the *FoR* Name;

G is the granularity of the ***FoR;***

D is the direction and biasing system used in the FoR (e.g., a Cartesian coordinate or four-dimensional frame that includes x, y, z, and time);

L is a set of directional labels used in the FoR (e.g., top, bottom, left, right); and

S is inherited ***FoRs.***

3.2.3.3 Granularity

Different grains may "smooth" objects, reduce "insignificant" objects to points, or map into spaces of different dimensionalities. Thus, spatial relations can be defined over entities in the abstraction spaces if they are indexed with the granularity of that space. Each grain definition must include a set of primitive mapping functions such as indistinguishability, adjacency, and membership.

Various relations may hold between grains. For example, we can say that Grain 1 refines Grain 2 if Grain 1 draws at least as many distinctions as Grain 2. A full theory of granularity should allow an agent to abstract away irrelevant detail from a problem by moving to coarser granularity. This is useful, for example, in maintaining compatibility between the design model of a part, its finite element model, its manufacturing model, and its assembly model. Granularity is very useful for dealing with time as either a discrete event or an interval. This allows one to compatibly represent a machining process as an interval at one granularity and as an instant at another. The initial definition for granularity is:

(Granularity N ∂ S) where,

N is granularity specification name,

∂ is a set of primitive mapping functions (e.g., adjacent, in, same), and

S is a set of inherited granularities.

3.2.3.4 Directional and Biasing Systems

A space may include metrics that need not necessarily be Cartesian because one may define metrics on spaces that do not have "straight line" concepts or define only partial orders over a range.

3.2.3.5 Reasoning

The shape-based reasoning strategy employed has three main stages: 1) a pre-attentive recognition stage that uses geometric algorithms to extract geometric and topological patterns, 2) an attentive stage that uses graph-search algorithms to match prototypes, and 3)

an introspective stage that uses artificial intelligence algorithms to assess the impact of these shapes.

3.2.3.5.1 Recognition Process

The recognition process involves three kinds of geometric and topological algorithms: 1) predicates, 2) constructions, and 3) properties.

3.2.3.5.1.1 Geometric and Topological Predicates

Examples of geometric predicates include coincident, coplanar, concave, convex, star-convex, collinear, perpendicular, parallel, inside, outside, on, above, below, interference, and same.

Examples of topological predicates include same, adjacent, articulation, and depression/protrusion.

Notably, the concept of *proximity* pervades the definition of all these predicates.

3.2.3.5.1.2 Geometric and Topological Constructions

The geometric and topological constructions are needed to extract geometric and topological information that is not explicitly encoded, or to use as representations that are better suited for property calculations. Examples of these constructions include convex hull, smallest enclosing box, smallest enclosing cylinder, smallest enclosing sphere, Voronoi diagram, union, intersection, difference, negation, projection, sweep, and binary space partition.

3.2.3.5.1.3 Geometric and Topological Properties

The geometric and topological properties that may be needed are length, area, volume, normal, and aspect ratio.

3.2.3.5.2 Prototype Matching

The prototype matching stage takes uninterpreted geometric and topological patterns and matches them to prototypes in a prototype knowledge base. These algorithms are of a

graph-search nature, such as connected components, articulation points, and minimum spanning trees.

3.2.3.5.3 Prototype Interpretation

The interpretation of prototypes involves the use of abductive reasoning. Abductive reasoning uses conjectures to explain observations and has a default prediction mechanism.

4.0 High Productivity Computer-Aided Design (HPCAD)

4.1 Introduction

The HPCAD Toolkit is a C++ library of object-oriented classes, methods, and functions that allows a programmer to create mechanical CAD/Computer-Aided Manufacturing (CAM)/CAE applications (electrical engineering CAD applications are not supported). Since HPCAD is not an application in its own right, it is not an executable program. It is a resource that a programmer would access to add high-level CAD/CAM/CAE functionality to a program.

4.2 Overview of the Functionality

The HPCAD Toolkit provides both high- and low-level CAD functionality for programming CAD/CAM/CAE applications. The Toolkit allows for the creation of either full-blown applications with a wide range of capabilities or specific, single-purpose applications.

The HPCAD Toolkit is not tied to any particular graphical user interface to allow for maximum portability and application interface control, nor is its implementation tied to any specific platform.

4.2.1 Key Components and Their Use

The HPCAD Toolkit is broken into the following five modules:

1. Solid Module – contains the classes, methods, and functions that relate to polyhedral CAD applications. The knowledge representations that are supported include the winged-edge and winged-triangle structures. The operations on these polyhedral solids include constructive solid geometry (CSG) operations (e.g., union, difference, intersection) and Euler operations (e.g., move-face,

extrude-face, lift-vertex). Also included are functions for translating between different polyhedral representations.

2. Curve and Surface Module – contains the classes, methods, and functions that relate to free-form surface CAD applications. The knowledge representations that are supported include nonuniform rational B-splines (NURBS) curves and surfaces, and Bézier curves and surfaces. Many surface creation methods are provided. Also included are functions for translating between different curve and surface representations.

3. Hybrid Object Module – contains the classes, methods, and functions that relate to combining polyhedral and surface objects. Both the Solid Module and Curve and Surface Module are necessary for hybrid object functionality. CSG operations (e.g., union, difference, intersection) are provided for operating between different curved surfaces or between a curved surface and a polyhedral solid. Also included are functions for translating between the hybrid object structure and different curved surface and polyhedral solid representations.

4. Geometric Display and Transformation Module – contains the classes, methods, and functions for generically displaying and transforming objects in three-dimensional (3D) space. Included are functions that control how objects are viewed, as well as those which transform objects (e.g., rotation, translation, scaling) in 3D space. These functions operate with virtually no direct links to a specific interface. These links must be provided by the programmer.

5. Support Functions Module – contains the classes, methods, and functions used to provide the basic functionality in the other modules. Included in this module are list and vector-related classes and functions. The lists behave similarly to lists in the Lisp programming language. The vector functions are very low-level geometric operations upon which the higher-level geometric functions in the other modules are built.

4.2.2 Underlying Concepts

The HPCAD Toolkit takes a function library approach to provide maximum functionality and flexibility. This gives Toolkit users the ability to fully customize applications and create them with whatever graphical user interface is desired.

The advantages of HPCAD are:

1. Open Architecture – allows users to write programs that can tie into existing applications (e.g., AutoCad).

2. Object-Oriented Approach – provides a clearer understanding of system behavior. It also provides enhanced reuse of code and increased maintainability.

3. Efficient Algorithms – provide for fast and robust execution. Every attempt has been made to provide for all special cases.

4. Generalized CSG (GCSG) – combining polyhedral solids and curved surfaces is highly desirable for many or most designs. This is also performed with little or no approximation, depending on the representation of resulting object.

5. Geometric Translations – translating between different geometric representations is extremely important for the sharing of product data; therefore, Toolkit provides many different functions for translating between winged-edge solids, winged-triangle solids, NURBS surfaces, Bézier surfaces, and boundary representations (BREPs).

5.0 Container Object System (COS)

5.1 Introduction

Modern engineers use a myriad of automated tools to aid them in the design process. A major thrust in current technology is to make these tools "intelligent." This requires vast amounts of stored knowledge and information. However, the question must be asked, "What is the best way to represent all this design knowledge that will allow it to be managed as an evolving asset?" Answering this question involves dealing with some tough issues. Unfortunately, none of the existing representations deal with all of them adequately.

COS proposes a new knowledge representation for CE design called *container objects*. These objects incorporate the best qualities of earlier representation efforts. The container object has been demonstrated as an effective solution to the problem of representing design knowledge within CE applications.

The task of representing knowledge has proven quite formidable. It seems that, every few years, some new scheme becomes the focus of industry knowledge engineers and computer scientists. Among these schemes can be found such proposed representations as rules, frames, objects, and even neural networks. Unfortunately, after users accumulate some experience with the latest representation, its shortcomings become all too obvious. Efforts to develop a single, unifying, knowledge representation scheme which effectively expresses many different (and changing) kinds of knowledge have not been entirely successful.

The CE paradigm places great demands on any candidate knowledge representation scheme. In particular, three issues must be addressed. First, this scheme must be powerful enough to express the concepts within a design domain, such as geometric, temporal, and hierarchical information. Second, it must not force its implementation to inhibit the simultaneous utilization and modification of the design of systems and subsystems by different users. Furthermore, in any useful implementation the effects of

any updates on the rest of the system must be computable and available to those who are designing the affected components. Finally, perhaps the most subtle demand CE places on knowledge representation schemes is that they must allow product descriptions to evolve naturally over time. Without this capability, the product design cannot be easily and efficiently modified and updated as the design process progresses. If the representation cannot capture all types of design knowledge or induces a communication bottleneck in any of its implementations, it will certainly not be adequate for a CE application.

Many proposed knowledge representation schemes have suffered from an inability to support product design data evolution. The source of this shortcoming is that representations of descriptive knowledge are almost always based on rigid structures that are not amenable to continual reorganization. Adding, modifying, and deleting descriptors is usually not allowed and is never endorsed. This rigidity prohibits the evolution of object descriptions, since the elements necessary to describe the final product usually are not known at the beginning of the design process. Unfortunately, nearly all current representation schemes suffer from the this problem.

The primary impetus for the research described herein is an overwhelming need for a design knowledge representation that can support the CE design process. This research is based on the observation that for any design knowledge representation scheme to be appropriate for CE applications, it must do the following:

1. express CE types of design knowledge,
2. allow simultaneous modification of design components, and
3. promote product design evolution.

No existing representation satisfies all these requirements. The working hypothesis is that a knowledge representation scheme will adequately support product design evolution if its representational constructs are defeasible (i.e., if they allow dynamic reorganization of descriptive knowledge). To this end, this research proposes the use of objects with extremely malleable descriptions (i.e., *container objects*) as the basis for design knowledge representation within CE applications.

The objective of this research is to devise a representation that satisfies the three CE design needs stated. Many existing representations satisfy the first two needs; only the promotion

of product design evolution remains elusive. This suggests that the best approach is to extend or reform some existing representation scheme that satisfies the first two so that it ultimately satisfies this remaining need. For this reason, the container object representation utilizes a great deal of prior work and research in the area of knowledge representation. In particular, it borrows heavily from earlier *object-oriented* representation schemes and from later derivatives called *composite object systems*.

5.2 Overview of the Functionality

5.2.1 Key Components and Their Use

COS is used by the other DKMS components as a common representation for design knowledge. It is composed of the following components.

1. Perspective Component – provides the user with the means to manipulate perspectives. It allows the user to define, retrieve, and modify perspectives, and is responsible for updating the container objects whose descriptions contain the perspective being modified. The external interface to this component is composed of the functions relative to the manipulation and modification of perspectives.

2. Container Object Component – provides the user with the means to manipulate container objects. It allows the user to create, destroy, retrieve, and modify container objects by adding and deleting perspectives to these objects. It also provides the means to initialize and modify container object attribute values. The external interface to this component is composed of the functions relative to the manipulation and modification of container objects.

3. Constraints Component - provides the user with the means to manipulate constraints. It is responsible for the propagation of the constraints defined in the system. The external interface to this component is composed of the functions relative to the definition and manipulation of constraining relations and constraints.

4. Generic Functions and Method Component – provides the user with the means to manipulate generic functions and methods. It contains the functions that allow the user to create and delete generic functions, and to create methods. The external interface to this component is composed of the functions relative to the manipulation and modification of generic functions and methods.

5. Persistent Storage Component – performs the functions in the COS associated with storing information in a nonvolatile medium. The

external interface to this component is composed of the persistent storage functions.

5.2.2 Underlying Concepts

COS is based on the container object representation scheme in which container objects represent real-world or conceptual entities and are described using *perspectives*. A perspective is a data organization structure similar to the notion of a *class* in conventional object-oriented languages. A perspective may be visualized simply as a set of essential properties which an object having this perspective must possess. Perspectives are composed of two types of descriptive elements: attributes that correspond to basic properties such as height or length, and components that correspond to the "parts" of an object and are used to create composite objects. Descriptive elements can be dynamically added or deleted from a perspective.

An object description is composed of perspectives. Each perspective corresponds to a different view of the object. For example, a bag can be viewed as a suitcase or a box. Perspectives can be dynamically added or deleted from an object definition.

Constraints are simply boolean expressions attached to containers which always have the value "TRUE" when evaluated. Some atomic elements of a constraint should be references to container attributes that can be the container to which the constraint is attached, or some component of the container to which the constraint is attached. A constraint may be defined at the perspective level so that all container objects described using the corresponding perspective are also constrained, or may be defined at the container-object level, so that only a specific container object is constrained.

To give COS the ability to act upon its elements, *methods* are provided. A method is a reusable sequence of instructions designed to work with specific data organization schemes (in this case, perspectives). The method must maintain information regarding its name, the data types (perspectives) upon which it may operate, place-holding identifiers called arguments, and, of course, the sequence of instructions called the *body*.

6.0 Model Design Support Environment (MDSE)

6.1 Introduction

MDSE provides a modeler with a framework for 1) producing accurate, timely models in a cost-effective manner; 2) leveraging the productivity of experienced modelers; 3) allowing inexperienced modelers to create more sophisticated models; and 4) preserving the knowledge of the expert modeler in such a way as to provide intelligently guided access.

6.2 Overview of the Functionality

6.2.1 MDSE within DKMS

MDSE is a "CAE tool" component of DKMS. This tool allows a designer to easily create performance prediction models for the systems being designed. In its final version, MDSE will allow a DKMS user to attach these analytic models to designs within HPCAD. In addition, it will provide a computer algebra facility for use by other DKMS applications.

6.2.2 Key Components and Their Use

MDSE is a tool for the computer-assisted development of computer models. It includes software for the creation, maintenance, and modification of engineering models and their numerically oriented software implementations. It combines several technologies to accomplish this goal, each of which constitutes a mature field in its own right.

The technologies in MDSE are:

1. Computer Algebra,
2. Constrain' . anagement,

3. Dimensional Analysis,

4. Automated Code Generation, and

5. Automated Documentation Generation.

The MDSE methodology emphasizes separation of knowledge acquisition and model development. To this end, the mathematical equations used for the analysis in a field are broken into two separate components: the state variables representing physical properties, and the equations representing relationships and correspondences. By creating this separation, it is hoped that reuse of engineering modeling knowledge and accuracy of the associated model implementation software can be improved. Through the MDSE interface, accumulation of the engineering modeling rationale and knowledge is simplified to make the system as unobtrusive as possible.

An additional advantage of this approach is that knowledge about modeling in a particular engineering domain can be separated from software implementation knowledge. This allows engineering users who do not understand the intricacies of programming to create sophisticated models in that domain and produce executable software implementations of those models. The modeling support is provided by an ability to reuse and tailor models created by experts in the domain. This support also includes the automatic generation of solution procedures for a constructed model, which allows the modeler to concentrate his/her attention in the area in which it will be most beneficial.

The knowledge acquired and stored in this manner can be passed on to future modelers. Much of the modeling done today is documented after the code is written, at which point the rationale for many decisions made in the model development process are lost.

6.2.2.1 Definition of Terms

MDSE maintains several knowledge bases concerning the field in which the modeling occurs. Using these bases, MDSE assists a user in creating, modifying, and maintaining numerically oriented software. MDSE has several concepts which, while fairly intuitive, must be understood by the user for the system to make sense.

6.2.2.1.1 Physical Properties

The most basic descriptor is the *physical property*. This property describes some feature of an object (such as its weight, length or density) in several significant ways. A physical property is specific to a class of objects; for instance, "average interior car temperature" is an example of a physical property for the object class "car." The list of attributes maintained for each characteristic contains, as a minimum, those attributes listed in the following table.

Name	A descriptive name of the physical property.
Symbol	A shorter descriptor for the physical property, limited to six characters, to be used in actual code generation.
Parent	The name of the physical property used to copy the default values initially. Physical properties initially copy all their default values from one other physical property. This is the name of that physical property.
Source	How the value was obtained (e.g., from direct measurement, a different model of the object, or reference material).
Range	The minimum and maximum values the physical property value can assume.
Unit	Unit of measurement in which the value is expressed (e.g., inch, kilogram, etc.).
Default	A default value for the physical property.
Type	Real, integer, complex, etc.
Constant	If the value is a constant that never changes, it is "true." If any software module ever solves for this value, it is not a constant; consequently, the value will be "false."
Granularity	The smallest interval that concerns the modeler. For instance, if we only care about the time to the nearest second, its granularity is one second.
Uncertainty	The extent to which the value of this property is unknown (e.g., if a temperature is measured to within .1 degree, because of instrument error its value is 1). Granularity and uncertainty differ in that granularity is a matter of intent, while uncertainty is a result of the physical limitations of the measuring instrumentation.
Citation	Reference noting the work from which this characteristic was taken, especially if a reference work.
Background	Where material can be found which describes this physical property.
Comment	Any important information not classified by the above categories.

By maintaining these features of a physical property, we can maintain the links to the information source on which the modeler is basing his/her decisions. This simplifies the process of updating the knowledge and, hence, the software. In addition, it can help a modeler who is not the original author to gain insight into the ideas used in the original model.

6.2.2.1.2 Relationships

MDSE refers to abstract mathematical statements as "Relationships." Examples of such relationships include 1) formulas from the physical sciences, 2) empirically derived relationships, and 3) mathematical definitions. Relationships are not specific to a particular object instance, but rather to the class of objects. That is, F = Ma is a relationship while "the force of bullet x913rq with mass 2 grams and acceleration of 350 m/sec^2" is not. In the process of modeling a specific product, determining how the generic relationships are mapped to physical properties of the parts of that product is a separate step in the performance model development and its software solution implementation. We will capitalize on this fact in the MDSE design to allow the abstraction of these relationships, thereby making them available for reuse. MDSE also supports the separation of relationships into the following two basic categories:

1. Absolute – implying that the relationship was derived from first principles, and
2. Empirical – meaning the relationship is the result of curve fitting data or some similar technique.

Relationships store the following information:

Name	A name for the relationship.
Type	Whether the equation is empirical or absolute.
Source	Whether this equation has been derived by the author, from a reference work, or has been developed in another program or model.
Citation	A reference from the literature.
Uncertainty	A percentage error inherent in the relationship. Applying only to empirical relationships, it is related to the error in the curve fit.
Background	Where material can be found which describes this physical property.
Limitations	Any restrictions that limit the application of the relationship.
Data Set	This information is kept for empirical relationships that refer to the set of data used to derive the relationship. This is used to automatically generate limitations to the circumstances in which the relationship can be applied, thus keeping modelers from misapplying empirical relationships because they are only defined within specific parameters related to the data set from which they are generated.
Comment	Any important information not classified by the above categories.

6.2.2.1.3 Programs

A modeler decides to create a piece of software; thus he/she defines a "Program." A program definition includes 1) a set of relationships, 2) a set of physical properties, and 3) a mapping between them. This mapping defines a set of *equations*. An equation is defined as a relationship with physical properties mapped to each of its variables. It is important to note that a relationship can be used to define an equation more than once.

Programs also have information stored with them that should guide the modeler in two aspects:

1. whether this program will meet a specific need, and
2. whether this program is defined such that it may contain information to help solve another problem.

To meet these goals, the following information is maintained for each program.

Problem Specification	A textual description of the problem this program is meant to solve.
Problem Example	An example of such a problem.
Solution Method	In general terms, the method that will be used to solve this problem. The emphasis for this description is to help the future modeler visualize the problem and its solution.
Solution Example	A step-by-step explanation of how the problem was solved.
Caveats	Limitations of the solution including operations or applications that should not be tried using this problem solution.
Approaches Not Tried	Approaches that were considered and an explanation of why the chosen solution seemed superior.
Comment	Any important information not classified by the above categories.

6.2.2.2 Method

This section discusses how MDSE works and how it can be fit into an organization.

6.2.2.2.1 Modeling

MDSE breaks modeling into two separate tasks: the collection of relationships and physical properties, and the organization of those into models. Separating these tasks allows a modeler to focus on the visualization and analysis for the model rather than the coding of the solution techniques. In addition, the knowledge bases can be maintained by persons without modeling experience, thereby reducing the cost of database maintenance.

When MDSE is in full use within an organization, it should be the central repository for all test data, measurements, and information on the domain that the organization needs to model. It is intended that MDSE have access to all this data in order to minimize the data collection effort.

6.2.2.2.2 Knowledge Base Support

Both relationships and physical properties are arranged in a hierarchy, with the default values of each being inherited; this reduces the effort required to enter information. Further, locating a relationship or physical property is simplified because of the logical structuring enforced by the hierarchy. For example, consider a relationship empirically developed from test data on the saturation pressure versus temperature in an air and water mixture. This relationship takes the form:

$$\text{Saturation-Pressure} = \exp(14.7276 - \frac{7259.2}{T + 391.2}).$$

This is an instance of the Antionne Equation (which relates the saturation pressure to the temperature for any two liquids, given three coefficients), an empirically derived relationship of the form:

$$\text{Saturation Pressure} = \exp(a - \frac{b}{T + c})$$

where a, b, and c are determined on the basis of the gases in the mixture.

MDSE might store such a relationship in the following form:

Saturation Pressure vs. Temperature Relations

Antionne's Equation

Antionne's Equation for water and air

This allows a user to enter data in the following manner, building up the information through inheritance.

Relationship 1	
Name	Saturation Pressure vs. Temperature Relations

Relationship 2	
Name	Antionne's Equation
Parent	Saturation Pressure vs. Temperature Relations
Type	Empirical
Source	Book
Citation	Reid and Sherwood, "Properties of Gases and Liquids," McGraw-Hill
Background	Thermodynamics Textbooks
Uncertainty	2%
Comment	Refer to Reid and Sherwood for more accurate temperature-pressure relations.
Form	Saturation pressure $= \exp\left(a - \frac{b}{T+c}\right)$

Relationship 3	
Name	Antionne's Equation for water and air
Parent	Antionne's Equation
Data Set	Unknown
Limitations	$T \leq 32.0$

The final definition is:

Name	Antionne's Equation for water and air
Parent	Antionne's Equation
Type	Empirical
Source	Book
Citation	Reid and Sherwood, "Properties of Gases and Liquids," McGraw-Hill
Background	Thermodynamics Textbooks
Uncertainty	2%
Comment	Refer to Reid and Sherwood for more accurate temperature-pressure relations.
Form	Saturation pressure $= \exp\left(a - \frac{b}{T+c}\right)$
Data Set	Unknown
Limitations	$T \leq 32.0$

This structure allows the modeler easy access to similar relationships. If the modeler felt the uncertainty inherent in this empirical relationship was too large, he/she could simply travel up the hierarchy to a level with the correct generic relationship, then look at other, more-specific relationships that had lower uncertainty.

6.2.2.3 Example

A scenario of use is the easiest way to understand how this process proceeds. The user in this scenario is a table manufacturer. Assume the user wants a program to calculate the weight of a table for shipping. He/she would first examine the database of past relationships for those relating to the problem, specifically by searching for the keyword "weight." The following relationship might be encountered:

$$\text{density} = \frac{\text{mass}}{\text{volume}}.$$

Density can be easily established (either through direct measurement or by examining reference works); therefore, a search should be performed for the word "volume." This would uncover the relationship:

length * depth * width = volume.

Since some relationships in this system have been defined, the physical properties in the system can be examined. The table has a total volume--the volume of the table top combined with the volume of the four legs. Further, the length, width, and depth of the top and legs is known. A physical property entry for each of these can be created; for the sake of brevity only the entry for the table leg length is shown here.

Name	table-leg-length
Source	Measurement
Range	(18.0 , 48.0)
Unit	Inch
Default	36.0
Type	Float
Constant	No
Granularity	.1
Confidence Factor	High
Citation	None
Comment	This property decides how tall the table will be.

Potential software solutions can be produced based on this information. First, the software production component of the system can be used to define a "program."

Problem Specification	Given the geometric description of a table, find its weight.
Problem Example	What is the weight of a cherry end table with a 24" x 36" x 1" top and 2" x 2" x 24" legs?
Solution Method	Find the volume of the table and the density; use them to find the weight.
Solution Example	Unknown.
Caveats	This model is only accurate with tables which have rectangular solids for legs and top.
Approaches Not Tried	More complex geometrics may more accurately predict the weight of complex tables.
Comments	A simple model for table weight.

This program gives a framework in which equations can be constructed by matching physical properties to relationships. The relationship for volume must be used multiple times; therefore, start by describing the table top as illustrated in Figure 5.

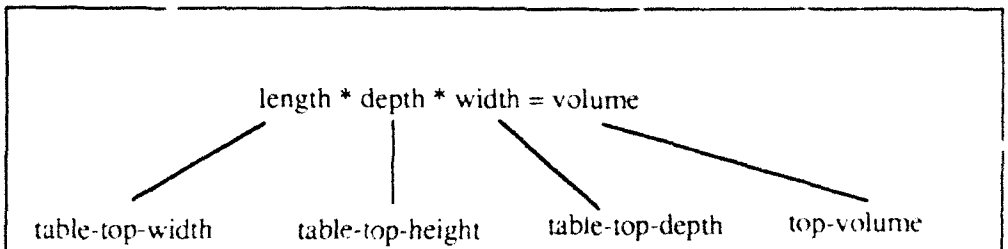


Figure 5. Table Top Equation Matching

This equation describes the relationship between the physical properties table-top-width, table-top-height, table-top-depth, and top-volume. The modeler must keep in mind that it is the act of matching physical properties to the variables within a relationship that transforms the relationship into an equation. The same relationship can be matched to the physical properties table-leg-height, table-leg-depth, table-leg-width, and leg-volume, as illustrated in Figure 6.

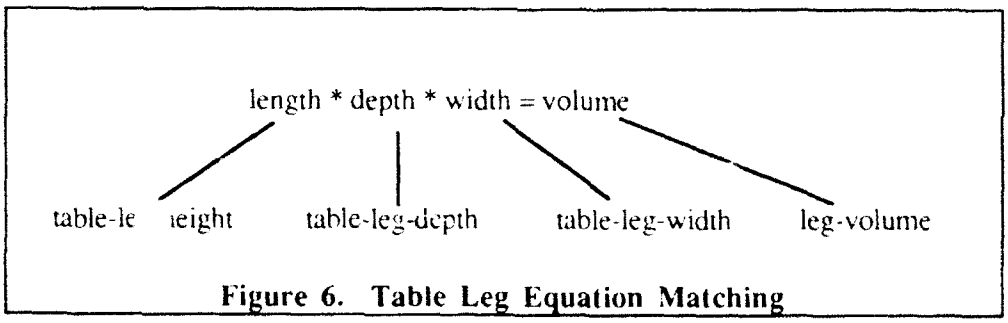


Figure 6. Table Leg Equation Matching

The relationship between the volume of the individual components and the overall volume has not yet been delineated. This relationship could have been defined from the start; however, it is also possible to define it at this point. Enter the relationship:

total-volume = top-volume + 4 * leg-volume

and match it as illustrated in Figure 7.

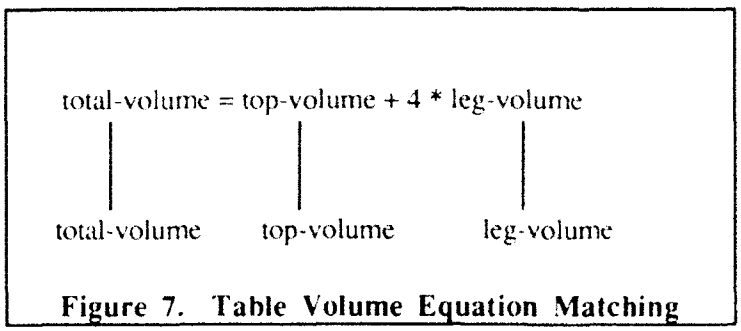


Figure 7. Table Volume Equation Matching

Software production can begin using these equations. To produce software, the programmer must know which physical properties are known at run time. This is done by selecting them from a list of all the physical properties in the defined equations. The physical property table-mass is not in any equation; therefore, an additional equation is required. From the list of relationships, the following equation, which has a relationship for mass, can be found:

$$\text{density} = \frac{\text{mass}}{\text{volume}}$$

Further, the density of the material from which the table made can be determined and the volume has already been computed. Therefore, this relationship can be used to find the mass of the table by matching it as shown in Figure 8.

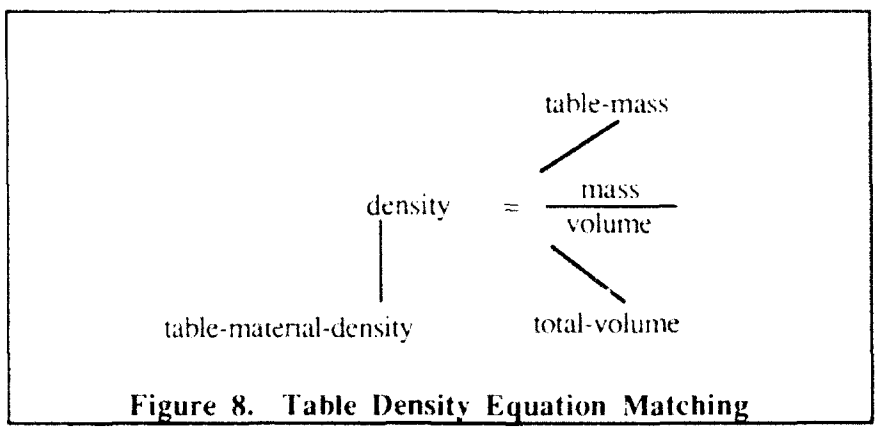


Figure 8. Table Density Equation Matching

Now the table-mass occurs in the list of possible unknowns, and table-top-width, table-top-height, table-top-depth, table-leg-height, table-leg-depth, table-leg-width, and table-material-density can be selected as the knowns. The system presents two sequences for solving these equations: one of which would allow solving for leg volume first, the other

would allow solving for top volume first. Since the ordering of these equations is irrelevant, the first solution sequence can be selected first.

Having received all necessary input from the user, the subprogram for solving this program is written to a file. The subprogram is documented with all the information the user has entered about the various physical properties and relationships. This allows someone who has the code, but not the MDSE, to understand the code and its purpose. The added benefit of having the documented subprogram in FORTRAN or C is the flexibility of moving the developed software to faster hardware.

In addition to software creation, MDSE has facilities for examining the relationships and programs that are developed. It can perform "sensitivity analysis" that allows the modeler to examine any physical property in terms of the other related properties in the program. A modeler often must try to distinguish between important features of the object being modeled and incidental characteristics that have little impact on the object as a whole. The sensitivity analysis can be used to determine when a physical property has such trivial effect.

7.0 Bibliography

[DKMS 91] Knowledge Based Systems, Inc. *DKMS Beta Test Training Document*. KBSI-DKMS-90-STR-01-1291-01, Volume 1, Contract F33615-90-C-0011, AL/HRGA, Wright-Patterson Air Force Base, OH. December 1991.

[KBSI 91a] Knowledge Based Systems, Inc. *Beta Test Training Document*, KBSI-DKMS-90-STR-01-0492-01, Volume 1, AL/HRGA, Wright-Patterson AFB, June, 1991.

[KBSI 91b] Knowledge Based Systems, Inc. *Software Design Document for the Container Object System*, KBSI-DKMS-90-SDD-04-0392-04, Volume 4 of 5, AL/HRGA, Wright-Patterson AFB, October, 1991.

[KBSI 91c] Knowledge Based Systems, Inc. *Software Design Document for the High Productivity CAD Toolkit*, KBSI-DKMS-90-SDD-03-0392-04, Volume 3 of 5, AL/HRGA, Wright-Patterson AFB, October, 1991.

[KBSI 91d] Knowledge Based Systems, Inc. *Software Design Document for the Model Design Support Environment*, KBSI-DKMS-90-SDD-05-0392-04, Volume 5 of 5, AL/HRGA, Wright-Patterson AFB, October, 1991.

[KBSI 91e] Knowledge Based Systems, Inc. *Software Design Document for the Integration Platform*, KBSI-DKMS-90-SDD-01-0392-04, Volume 1 of 5, AL/HRGA, Wright-Patterson AFB, October, 1991.

[KBSI 91f] Knowledge Based Systems, Inc. *Software Design Document for the Shape-based Design Knowledge Representation and Reasoning System*, KBSI-DKMS-90-TR-02-0492-01, Volume 2 of 5, AL/HRGA, Wright-Patterson AFB, October, 1991.

[KBSI 91g] Knowledge Based Systems, Inc. *Software Requirements Specification for the High Productivity CAD Toolkit*, KBSI-DKMS-90-SRS-03-0392-01, Volume 3 of 5, AL/HRGA, Wright-Patterson AFB, July, 1991.

[KBSI 91h] Knowledge Based Systems, Inc. *Software Requirements Specification for the Container Object System* KBSI-DKMS-90-SRS-04-0392-01, Volume 4 of 5, AL/HRGA, Wright-Patterson AFB, July, 1991.

[KBSI 91i] Knowledge Based Systems, Inc. *Software Requirements Specification for the Model Design Support Environment* KBSI-DKMS-90-SRS-05-0392-02, Volume 5 of 5, AL/HRGA, Wright-Patterson AFB, July, 1991.

[KBSI 91j] Knowledge Based Systems, Inc. *Software Requirements Specification for the Integration Platform*, KBSI-DKMS-90-SRS-01-0392-01, Volume 1 of 5, AL/HRGA, Wright-Patterson AFB, July, 1991.

[KBSI 91k] Knowledge Based Systems, Inc. *Software Requirements Specification for the Shape-based Design Knowledge Representation and Reasoning System* KBSI-DKMS-90-SRS-02-0492-01, Volume 2 of 5, AL/HRGA, Wright-Patterson AFB, July, 1991.

[KBSI 91l] Knowledge Based Systems, Inc. *Container Object System (COS) Programmer's Manual,* KBSI-DKMS-90-SUM-04-0392-01, Volume 4 of 5, AL/HRGA, Wright-Patterson AFB, May, 1991.

[KBSI 91m] Knowledge Based Systems, Inc. *High Productivity CAD (HPCAD) Programmer's Manual,* KBSI-DKMS-90-SUM-03-0392-01, Volume 3 of 5, AL/HRGA, Wright-Patterson AFB, May, 1991.

[KBSI 91n] Knowledge Based Systems, Inc. *Integration Platform (IP) User's Manual,* KBSI-DKMS-90-SUM-01-0392-01, Volume 1 of 5, AL/HRGA, Wright-Patterson AFB, May, 1991.

[KBSI 91o] Knowledge Based Systems, Inc. *Model Design Support Environment (MDSE) User's Manual,* KBSI-DKMS-90-SUM-05-0392-01, Volume 5 of 5, AL/HRGA, Wright-Patterson AFB, May, 1991.

[KBSI 91p] Knowledge Based Systems, Inc. *Shaped-Based Design Knowledge Representation and Reasoning (SBDKRR) User's Manual,* KBSI-DKMS-90-SUM-02-0492-01, Volume 2 of 5, AL/HRGA, Wright-Patterson AFB, May, 1991.

[KBSI 91q] Knowledge Based Systems, Inc. *Technology Impact Report,* KBSI-DKMS-90-STR-01-0292-03, Volume 1 of 1, AL/HRGA, Wright-Patterson AFB, February, 1992.

☆U.S.Government Printing Office: 1993 - 750-061/60240